

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

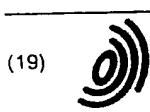
Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problem Mailbox.**



(19) Europäisches Patentamt
European Patent Office
Office européen des brevets



(11) EP 1 074 911 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication: (51) Int. Cl.⁷: G06F 9/44

07.02.2001 Bulletin 2001/06

(21) Application number: 00305804.7

(22) Date of filing: 10.07.2000

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU

MC NL PT SE

Designated Extension States:

AL LT LV MK RO SI

(30) Priority: 20.07.1999 US 357738

(71) Applicant: Xerox Corporation
Rochester, New York 14644 (US)

(72) Inventors:

• Kiczales, Gregor J.

Palo Alto, CA 94306 (US)

• Lamping, John O.

Los Altos, California 94024 (US)

- Lopes, Cristina V.
San Francisco, CA 94132 (US)
- Hugunin, James J.
Sunnyvale, California 94086 (US)
- Hilsdale, Erik A.
Bloomington, IN 47401 (US)
- Boyapati, Chandrasekhar
Cambridge, Massachusetts 01239 (US)

(74) Representative:
Skone James, Robert Edmund et al
GILL JENNINGS & EVERY
Broadgate House
7 Eldon Street
London EC2M 7LH (GB)

(54) Aspect-oriented programming

(57) The concept of a general purpose aspect is introduced where an aspect transparently forces cross-cutting behavior on object classes and other software entities. A reusable aspect is further described for use as part of an aspect library (310).

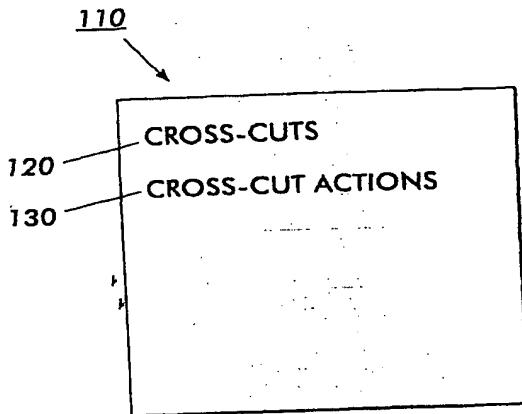


FIG. 1

EP 1 074 911 A2

Description

[0001] The present invention relates generally to creating software, and more particularly to aspect-oriented programming and aspect libraries.

[0002] Procedural languages such as for Fortran, Pascal, and C are useful for defining programs where the execution is straightforward, beginning at a starting point and executing in a stepwise manner to an end. In this model, design issues can be addressed by units of contiguous program execution. Deviations from the straightforward path are provided by function calls which allow program execution to jump from the main routine to the subroutine, and back again to the main routine. The use of subroutines allows for programming efficiency for implementing common routines; however, with programs becoming increasingly more complicated, and the number of common routines also growing, programs written in procedural languages are becoming increasingly complicated and difficult to maintain.

[0003] With modern computer programs becoming increasingly long and complex creations which may have many millions of lines of code, the concept of modularity is becoming increasingly important in the development of software. With a modular approach, the various functions of a computer program may be separated into modules which various programmers can work on independently. One popular programming paradigm that embodies the concept of modularity is that of object-oriented programming (OOP).

[0004] The central idea behind object-oriented programming is the object model, where all programs are structured as collections of interrelated objects, each of which represents an instance of some class in a hierarchy of object classes.

[0005] Object-oriented programming involves defining, creating, using, and reusing "objects", which can be used to model ideas and things in terms of their features (data) and behaviors (methods). Each object is a self-contained software element including data and methods for operating on the data. Objects are created by defining object classes from which objects are created, or "instantiated". The object classes are templates for creating objects. Each object created from a particular object class includes all the data and methods of the object class, as well as data and methods from its superclasses, and different objects of the same object class may be used for different purposes. Common object-oriented programming languages include Smalltalk, C++, and Java. Object classes for implementing commonly-encountered issues are often packaged as class libraries which developers can call upon, rather than having to re-create new object classes.

[0006] Other, non-OOP approaches are also commonly used, such as embodied in procedural programming languages and functional programming languages.

[0007] When design features may be cleanly divided among distinct elements, these approaches provide good support for programming those features in a modular way. However, these approaches fail to provide the proper support in certain situations, such as those involving shared resources, error handling, or other systemic issues where the same or similar functionality affects or is affected by many different elements.

[0008] The reason why these approaches are insufficient is that those issues cross-cut the primary modularization of the systems. Cross-cutting occurs when some particular concern depends on and/or must affect parts of the implementation of several of the functional modules of the system. Many cross-cuts are not weaknesses of the designs; they are a natural and unavoidable phenomena in complex systems, and they are the basis for the concept of "aspect".

[0009] Implementing those cross-cutting concerns in traditional programming languages, even object-oriented ones, typically requires scattering bits of code throughout the program, resulting in code that is tangled.

[0010] For example, non-trivial performance optimizations tend to cross-cut many modules when writing programs using traditional programming languages. So, for example, using Lisp or Java, the implementation of a cross-cutting performance optimization ends up affecting the non-optimized, modular implementation of the components in ways that are all but obvious, resulting in code that is tangled and in loss of the optimization abstraction.

[0011] In accordance with one aspect of the present invention, a computer system has a processor, a memory, and a software entity comprising either a program body or one or more object classes, the software entity defining an overall functionality of a software application program, and an aspect-oriented programming environment supporting an aspect that implements concerns that cross-cut the overall functionality of the software entity, the aspect comprising:

- 50 a) a cross-cut that comprises a point in the execution where cross-cutting behavior is to be included; and
- b) a cross-cut action comprising a piece of implementation associated with the cross-cut, the piece of implementation comprising computer-executable code that implements the cross-cutting behavior;
- c) wherein the cross-cut action transparently forces the cross-cutting behavior on the software entity.

[0012] An aspect is a concern that cross-cuts the primary modularization of a software system. An aspect-oriented programming language extends traditional programming languages with constructs for programming aspects. Such constructs can localize the implementation of cross-cutting concerns in a small number of special program modules, rather than spreading the implementation of such concerns throughout the primary program modules.

[0013] In order to capture the cross-cutting nature of aspects, such special program modules break the traditional rules of encapsulation in principled ways. They can affect the implementation of the primary modules without the explicit consent of those primary modules; further, they can do that for several primary modules simultaneously.

5 [0014] Aspect oriented programming (AOP) extends the expressive facilities available to the programmer, so that many design decisions can be expressed locally. The AOP programmer writes the base program in a traditional programming language, and also writes pieces of aspect code, each of which affects executions that are described in some parts of the base program.

10 [0015] In such a manner, aspect code can localize the implementation of some design patterns in a few modules, rather than spreading the fields and methods of those patterns throughout the classes, and can capture the tracing, debugging and instrumentation support for a complex system in a few modules, capture error handling protocols involving several classes in a single module, and capture resource sharing algorithms involving several classes in a single module, rather than as multiple code fragments tangled throughout the classes.

15 [0016] The special program modules for programming aspects enable this by cross-cutting the modularity of classes in principled ways. So one of those special program modules can affect the implementation of several classes (or several methods within a single class) in a clear, principled way. Aspect-Object interaction differs from Object-Object interaction and other traditional programming paradigms in that with the traditional approaches, all behaviors of the objects are encapsulated in the objects themselves, either as a direct implementation in the object class definition, as a request encoded in the object class definition to use the behaviors of other objects (e.g., a method call), or as a request in the object class definition to reuse the implementations of other object classes (e.g. through inheritance).

20 [0017] Thus, in these traditional approaches, all control of an object's behavior lies with the object itself. In the AOP environment, on the other hand, a part of the object's behavior can be defined in an aspect outside of the object without the object having to request the behavior in any way. Thus, it can be said that a part of the object's behavior is transparently forced on the object by the aspect. Moreover, aspects have a more global effect in that one aspect can force its behavior on multiple objects, possibly of different classes.

25 [0018] The paradigm of Aspect-Oriented Programming was first introduced in Gregor Kiczales et al., *Aspect-Oriented Programming in Proceedings of the European Conference on Object-Oriented Programming (ECOOP 97)*, June 1997 ("Kiczales"), which is hereby incorporated by reference. In Kiczales, a new unit of software modularity, called an aspect, was provided that appears to provide a better handle on managing cross-cutting concerns.

30 [0019] In Kiczales, only highly domain-specific aspect-oriented systems had been developed. There was also a goal of developing a general purpose AOP mechanism. At the time of the prior art, it was unknown how to generalize from the very specific examples of AOP to conceive of the necessary abstractions in order to create a general model. Thus, the aspect-oriented programming of the prior art was a hypothetical paradigm having the goal of providing a clean separation between and among components and aspects.

35 [0020] In Kiczales, four highly domain-specific aspect-oriented systems had been developed: AML, RG, ETCML, and D. For each of these cases, a new language had to be created, and a new compiler had to be implemented. While each of these languages included a notion of an "aspect", none of these aspects could be used with any of the languages for the other three because the foundation principles underlying a general-purpose aspect language were not known. Thus, each time a new aspect-oriented application is to be created, a designer would first have to create a new language, and implement a new compiler before actually developing the software application.

40 [0021] The present invention presents a general-purpose aspect-oriented programming paradigm. The environment embodying the present invention is called AspectJ, and avoids the shortcomings of prior art programming paradigms. AspectJ is an extension to the Java programming language. In AspectJ, object code is encapsulated in Java classes, and aspect code is encapsulated in special program modules called "aspects".

45 [0022] The generalization embodied in the present invention is based on the realization that cross-cuts can be represented by capturing points in the computation, such as message sends, method definitions, object accesses, paths in sequences of calls, etc.

[0023] To improve the usefulness of aspects in simplifying software development, a library aspect is introduced which allows for the abstract implementation of AOP.

50 [0024] Figure 1 illustrates an aspect according to an example of the present invention; Figure 2 is a flowchart depicting a method for creating a software program according to an example of the present invention; Figure 3 depicts an aspect library; and, Figure 4 is a block diagram of a computer system.

55 [0025] In this example, the method is implemented on a computer system 400 (Figure 4) having a processor 410 and a memory 420 including a software entity 430.

[0026] The present invention is a system for facilitating the development of software programs using aspect-oriented programming.

ented techniques. In accordance with the present invention, the system of the present invention executes on a computer, such as a general purpose personal computer of a type well-known in the art.

5 [0025] Referring to Figure 1, an aspect 110 according to the present invention provides a general way of implementing cross-cutting behavior. To do so, the aspect includes a cross-cut 120 and a cross-cut action 130. These cross-cuts can be concrete or abstract, as will be described in greater detail in conjunction with the discussion of aspect libraries, below. The cross-cut includes points in the computation where the cross-cutting action is to occur.

[0026] A point in the computation of a program is an action of the computation corresponding to a feature of the programming language in which the program is developed.

10 [0027] Programming languages have features that describe kinds of actions to perform. A program written in the programming language uses these features to call for specific such actions. When the program is executed, the implementation dynamically carries out a sequence of these actions in accordance with the program's instructions. Those are referred to as points in the computation.

15 [0028] Examples of points in the computation include reading a variable, a method invocation, the start of the execution of a method, the return from a method invocation, instantiation of an object, test of a condition, the raising of an exception, the capture of an exception, etc.

20 [0029] Being able to refer to an intended set of points in the computation is a key technical requirement for having a general-purpose aspect language. But such a goal is not trivial to achieve, because many of those sets of points do not have an explicit embodiment in the source code of the program. For example, objects are instantiated dynamically at run-time. Therefore denoting a method invocation on a specific object cannot be done through the use of a simple name, but must be done through denoting a variable that refers to that object and/or through enumerating properties of that object.

25 [0030] A general-purpose aspect language provides support for identifying sets of points of interest in the computation and for referring to them in the source code. By providing this support, the programmer can then associate one single piece of behavior with many different points in the computation.

30 [0031] A cross-cut action is a piece of implementation (i.e., code) that is to be executed on a cross-cut. These cross-cut actions may be implemented using common programming techniques and common programming language constructs.

35 [0032] Figure 2 illustrates the application, of AOP in developing and/or modifying software. AOP can be applied to any previously developed software application for which source code is available. This can be useful for applying such functionality as system monitoring or tracing, both of which are described in greater detail in our copending US patent application entitled ASPECT-ORIENTED SYSTEM MONITORING AND TRACING filed on 20 July 1999, which is incorporated by reference into the present specification. AOP can also be applied to previously developed software to add new functionality which cross-cuts the primary functionality of the software application.

40 [0033] In developing new software, AOP can be used to software monitoring and tracing as described above, and can also be used to implement any other cross-cutting issues for the software application. A benefit of AOP is that the defining of aspects may be done concurrently with the development of the software entities defining the primary functionality of the software application.

45 [0034] In either developing or modifying, primary functionality for the software is created as one or more object classes or other program bodies. Cross-cutting functionality is added to the software through the use of an aspect. Aspects implementing cross-cutting behavior are defined at step 210. Then, in order to have the aspects force their behavior on the object classes, the aspects are compiled, at step 220, along with the object classes.

50 [0035] In a present embodiment of the invention, a compiler for performing the compilation of step 220 takes the source files for the aspects and object classes, and produces intermediate object classes that are then compiled into byte codes. Those skilled in the art will readily appreciate that other ways of performing the compilation are possible without departing from the spirit of the invention. For example, the compilation might be done at the byte code level.

[0036] Tables 1 through 3, below, contain source code for an object class, and two aspect definitions, respectively.

```

----- File: Point.java -----
5
import java.io.*;
import java.util.*;

10
class Point {
    private int _x = 0;
    private int _y = 0;

    Point() {}

15
    void set (int x, int y) {
        _x = x; _y = y;
    }

20
    void setX (int x) { _x = x; }
    void setY (int y) { _y = y; }

    int getX() { return _x; }
    int getY() { return _y; }
}

```

Table 1 -Source code for object class Point

25

30

35

40

45

50

55

```

----- File: ShowAccesses.java -----
30
import java.io.*;

35
aspect ShowAccesses {
    public static void main(String[] args) {
        Point p1 = new Point();
        Point p2 = new Point();
        Point p3 = new Point();

        for (int i = 0; i <= 100; i++ ) {
            p1.set(i, i);
            p2.set(i, i);
            p3.set(i, i);
        }
    }

45
    advise Point(*), Line(*) {
        static before {
            System.out.println("C");
        }
    }
    advise * Point.getX(*), * Point.getY(*),
}

```

```

5      * Line.getX1(*), * Line.getY1(*),
      * Line.getX2(*), * Line.getY2(*) {
  static before {
    System.out.println("R");
  }
}
advise * Point.set(*), * Point.setX(*), *
Point.setY(*),
      * Line.set(*), * Line.setX1(*), *
Line.setY1(*),
      * Line.setX2(*), * Line.setY2(*) {
  static before {
    System.out.println("W");
  }
}

```

Table 2 - Source code for aspect ShowAccesses

20

25

```

----- File: AutoReset.java -----
import java.io.*;

aspect AutoReset {
  public static void main(String[] args) {
    Point p1 = new Point();
    Point p2 = new Point();
    Point p3 = new Point();

    AutoReset a1 = new AutoReset();
    a1.addObject(p1);
    a1.addObject(p2);
    a1.addObject(p3);

    for (int i = 0; i <= 100; i++ ) {
      p1.set(i, i);
      p2.set(i, i);
      p3.set(i, i);
    }
  }

  int count = 0;
  advise * Point.set(..), * Point.setX(..), *
Point.setY(..) {
  after {
    if (++count >= 100) {
      System.out.println("Reseting " +
thisObject + ".");
      count = -1;
      x = 0;
    }
  }
}

```

55


```
5           (AutoReset)_thisAspect;
6           {
7               if (++thisAspect.count >= 100)  {
8
9                   java.lang.System.out.println("Reseting " + this +
10                  ".");
11
12                   thisAspect.count = -1;
13                   _x = 0;
14                   _y = 0;
15
16               }
17
18           }
19
20       }
21
22   }
23
24
25   /*
26   * This method contains the original method
27   body for
28   * void setX(int x)
29   * This method is called from the original
30   method.
31   */
32
33   protected final void Point$setX(int x) {
34       _x = x;
35   }
36
37   /*
38   * The body of this member was replaced by
39   aspectj
40   * At the core of this code is a call to
41   * protected final void Point$setX(int x)
42   * which holds the original method body.
43   * Around this call is the code for all advise
44   * weaves that apply to this member.
45   */
46
47
48
49
50
51
52
53
54
55
```

```

5      void setX(int x) {
6      {
7          /*
8             * Generated by aspectj
9             * This implements the before advice *
10            setX()
11            * weave from the aspect ShowAccesses.
12            (ShowAccesses.java:27)
13            */
14            {
15                java.lang.System.out.println("W");
16            }
17            Point$setX(x);
18            /*
19             * Generated by aspectj
20             * This implements the after advice *
21            setX()
22            * weave from the aspect AutoReset.
23            (AutoReset.java:24)
24            */
25            {
26                java.util.Enumeration _enumeration =
27                _aspects.elements();
28                while (_enumeration.hasMoreElements()) {
29                    java.lang.Object _thisAspect =
30                    _enumeration.nextElement();
31                    if (_thisAspect instanceof AutoReset)
32                    {
33                        final AutoReset thisAspect =
34                        (AutoReset)_thisAspect;
35                        {
36                            if (++thisAspect.count >= 100) {
37
38                                java.lang.System.out.println("Reseting " + this +
39                                ".");
40                                thisAspect.count = -1;
41                                x = 0;
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
828
828
829
829
830
831
832
833
834
835
836
837
838
838
839
839
840
841
842
843
844
845
846
847
847
848
849
849
850
851
852
853
854
855
856
857
858
858
859
859
860
861
862
863
864
865
866
867
867
868
869
869
870
871
872
873
874
875
876
877
877
878
878
879
879
880
881
882
883
884
885
886
886
887
887
888
888
889
889
890
891
892
893
894
894
895
895
896
896
897
897
898
898
899
899
900
901
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
16
```

```

5
6         _y = 0;
7     }
8     }
9     }
10    }
11
12    }
13
14    }
15
16    /**
17     * This method contains the original method
18     body for
19     * void setY(int y)
20     * This method is called from the original
21     method.
22     */
23     protected final void Point$setY(int y) {
24
25         _y = y;
26     }
27
28
29    /**
30     * The body of this member was replaced by
31     aspectj
32     * At the core of this code is a call to
33     * protected final void Point$setY(int y)
34     * which holds the original method body.
35     * Around this call is the code for all advise
36     * weaves that apply to this member.
37     */
38     void setY(int y) {
39
40         /*
41          * Generated by aspectj
42          * This implements the before advice *
43
44         setY()
45          * weave from the aspect ShowAccesses.
46
47         (ShowAccesses.java:27)

```

```

5
6     */
7     {
8         java.lang.System.out.println("W");
9     }
10    Point$setY(y);
11    /*
12     * Generated by aspectj
13     * This implements the after advice *
14
15    setY()
16        * weave from the aspect AutoReset.
17        (AutoReset.java:24)
18        /*
19        {
20            java.util.Enumeration _enumeration =
21            _aspects.elements();
22            while (_enumeration.hasMoreElements()) {
23                java.lang.Object _thisAspect =
24                _enumeration.nextElement();
25                if (_thisAspect instanceof AutoReset)
26                {
27                    final AutoReset thisAspect =
28                    (AutoReset)_thisAspect;
29                    {
30                        if (++thisAspect.count >= 100) {
31
32                            java.lang.System.out.println("Reseting " + this +
33                            ".");
34
35                            thisAspect.count = -1;
36                            _x = 0;
37                            _y = 0;
38                        }
39                    }
40                }
41            }
42        }
43    }
44}
45
46
47
48
49
50
51
52
53
54
55

```

```

5
  /*
   * This method contains the original method
10 body for
   * int getX()
   * This method is called from the original
method.
15  /*
   * protected final int Point$getX() {
   *     return _x;
   * }
20
   /*
25   * The body of this member was replaced by
aspectj
   * At the core of this code is a call to
   * protected final int Point$getX()
   * which holds the original method body.
   * Around this call is the code for all advise
   * weaves that apply to this member.
30  /*
35   int getX() {
   *     int thisResult;
   *     /*
   *      * Generated by aspectj
   *      * This implements the before advice *
40   *      * weave from the aspect ShowAccesses.
   *      * (ShowAccesses.java:20)
   *      */
45   *     {
   *       java.lang.System.out.println("R");
   *     }
   *     thisResult = Point$getX();
50   }
   *     return thisResult;
   */

```

55

```

5
}
10
15
20
25
30
35
40
45
50
55
}
/*
 * This method contains the original method
body for
 * int getY()
 * This method is called from the original
method.
 */
protected final int Point$getY() {
    return _y;
}

/*
 * The body of this member was replaced by
aspectj
 * At the core of this code is a call to
 * protected final int Point$getY()
 * which holds the original method body.
 * Around this call is the code for all advise
 * weaves that apply to this member.
 */
int getY() {
    int thisResult;
    {
        /*
         * Generated by aspectj
         * This implements the before advice *
getY()
         * weave from the aspect ShowAccesses.
(ShowAccesses.java:20)
        */
        {
            java.lang.System.out.println("R");
        }
        thisResult = Point$getY();
    }
}

```

```

    return thisResult;
}

5
protected java.util.Vector _aspects = new
java.util.Vector();
10  public java.util.Vector getAspects() {
    return _aspects;
}
15
}

```

Table 4 - Intermediate code for object class Point

20

25

30

35

40

45

50

```

----- File: ajworkingdir\ShowAccesses.java -----
-----  

import java.io.*;
class ShowAccesses extends java.lang.Object {
    public static void main(java.lang.String[]
args) {
    Point p1 = new Point();
    Point p2 = new Point();
    Point p3 = new Point();
    for (int i = 0; i <= 100; i++) {
        p1.set(i, i);
        p2.set(i, i);
        p3.set(i, i);
    }
}
}

```

Table 5 - Intermediate code for aspect ShowAccesses

55

```
----- File: ajworkingdir\AutoReset.java -----
-----
5
import java.io.*;
class AutoReset extends java.lang.Object {
10    public static void main(java.lang.String[]
args) {
    Point p1 = new Point();
    Point p2 = new Point();
    Point p3 = new Point();
    AutoReset a1 = new AutoReset();
    a1.addObject(p1);
    a1.addObject(p2);
20    a1.addObject(p3);
    for (int i = 0; i <= 100; i++) {
        p1.set(i, i);
        p2.set(i, i);
        p3.set(i, i);
    }
30}
    int count = 0;
    private java.util.Vector _objects = new
35    java.util.Vector();
    public java.util.Vector getObjects() {
        return _objects;
    }
40
    public void addObject(Point object) {
        if (!_objects.contains(object)) {
            object.getAspects().addElement(this);
45            _objects.addElement(object);
        }
    }
50
    public void removeObject(Point object) {
        object.getAspects().removeElement(this);
    }
}
-----
```

55

```

    _objects.removeElement(object);
}

}

```

Table 6 - Intermediate code for aspect AutoReset

10

[0038] By examining the source code examples in Tables 1 through 3, and comparing them with the intermediate code examples in Tables 4 through 6, and especially the comments in Table 4, one of ordinary skill in the art will understand the transformations necessary for implementing a compiler according to the present invention.

15

Aspect Libraries

[0039] With the increasing complexity of today's systems, and the increasing demand for user-friendliness, modern software programs require a vast number of elements. Many of these elements, especially those concerned with user interfaces, and those addressing commonly faced issues, will be used over and over, both within the same software program, and for other programs. Rather than creating a new solution each time an issue reappears, a general routine for implementing the solution may be accessed from a library. Typically a software entity, which may be one or more program bodies or one or more object classes, embodying the primary functionality of the software program will include calls to a library routine. In object-oriented programming, for example, object class libraries are popular resources for developing software applications because they allow the developer to concentrate on writing code for new issues rather than having to re-create solutions that have previously been addressed by other developers in other applications.

[0040] Referring to Figure 3, an aspect library 310 is shown including a number of library aspects 312. Library aspects may be concrete or abstract. A concrete library aspect is the same as the aspect 110 illustrated in Figure 1, only packaged as part of an aspect library. It is described as being concrete because all the cross-cuts in it refer to concrete points in the computation, i.e. all the cross-cuts are bound to specific places in object classes. In a present embodiment of the invention, the concrete aspect extends the abstract library aspect through inheritance, but other ways of implementing this extension can be practiced without departing from the spirit of the invention.

[0041] The power of an aspect library lies in library aspects which are provided as abstract aspects in that their implementations can be reused for several different sets of points in the computation. An exemplary abstract library aspect 315 includes crosscut actions as in aspect 110 if Figure 1, but the crosscut 317 of the abstract library aspect acts as a placeholder rather than being specified as a particular point in the execution.

[0042] When use of an abstract library aspect is desired, it is bound to concrete points in the execution of an object (not shown) created from object class 320 by a concrete aspect 330 defined by the developer. In the concrete aspect, the developer uses the crosscut to define bindings 335 between the names of the cross-cuts 317 in the library aspect 315 to concrete points in the execution 325 of the object class 320.

[0043] In a present embodiment of the invention, a compiler for performing the compilation of abstract cross-cuts 317 of an abstract library aspect 315 takes the source files for the object classes and the concrete and abstract library aspects, and produces intermediate object classes that are then compiled into byte codes. Those skilled in the art will readily appreciate that other ways of performing the compilation are possible without departing from the spirit of the invention. For example, the composition might be done at the byte code level, and might make use of the extensibility of Java's byte code format, e.g., tags.

[0044] Tables 7 through 9 contain source code for an abstract library aspect, a concrete aspect, and an object class, respectively.

50

55

```

----- MethodDebug.java -----
5 aspect MethodDebug {
    abstract crosscut debugTypes;
    crosscut allMethods: * * (...) on @debugTypes;

10 static advise @allMethods(..) {
    before {
        System.out.println("about to call
15    "+thisJoinPoint.methodName);
    }
    after {
        System.out.println("have called
20    "+thisJoinPoint.methodName+
                    ", returns
25    "+thisResultObject);
    }
}
30

```

Table 7 - Source code for abstract library aspect
MethodDebug

```

35 aspect DebugMyClass extends MethodDebug {
    crosscut debugTypes: on MyClass;
}
40

```

Table 8 - Source code for concrete aspect DebugMyClass

```

45 class MyClass {
    public void method() {
        System.out.println("MyClass");
    }
    public void bar(int x) {
}
50

```

```

    System.out.println("bar(" + x + ") on MyClass");
}
}

```

5 Table 9 - Source code for object class MyClass

10 [0045] Tables 10 through 12 contain source code for three intermediate object classes corresponding to the two aspects and the object class in Tables 7 through 9, respectively. Table 12 also contains comments which are helpful in describing the transformations. These the intermediate codes generated by the preprocessing phase of the compiler.

```

----- ajworkingdir\MethodDebug.java -----
-----
class MethodDebug extends java.lang.Object {
}

```

20 Table 10 - Intermediate code for object class
25 MethodDebug

```

----- ajworkingdir\DebugMyClass.java -----
-----
class DebugMyClass extends MethodDebug {
}

```

30 Table 11 - Intermediate code for object class
35 DebugMyClass

```

----- ajworkingdir\MyClass.java -----
-----
class MyClass extends java.lang.Object {
    protected final void MyClass$method$signature()
}

```

40

50

55

```

5
{
    java.lang.System.out.println("MyClass");
}

10
public void method() {
{
    /*
     * Generated by aspectj
     * This implements the before advice * *()
     * weave from the aspect MethodDebug.
(LibTest.java:13)
    */
{
    java.lang.System.out.println("about to
call " + "method");
}
25
    MyClass$method$signature();
/*
     * Generated by aspectj
     * This implements the after advice * *()
     * weave from the aspect MethodDebug.
(LibTest.java:16)
    */
{
    java.lang.System.out.println("have called
" + "method" + ", returns " +
((java.lang.Object)null));
}
35
}
}

40
}
}

45
protected final void MyClass$bar$signature(int
x) {
    java.lang.System.out.println("bar(" + x + ")"
on MyClass");
}
50
}

```

```

5
public void bar(int x) {
10
    /*
     * Generated by aspectj
     * This implements the before advice * */
15
    (LibTest.java:13)
    /*
     */
20
    {
        java.lang.System.out.println("about to
call " + "bar");
        }
25
        MyClass$bar$signature(x);
        /*
         * Generated by aspectj
         * This implements the after advice * */
         * weave from the aspect MethodDebug.
30
        (LibTest.java:16)
        /*
         */
35
        {
            java.lang.System.out.println("have called
" + "bar" + ", returns " +
((java.lang.Object)null));
            }
        }
40
    }
}

```

Table 12 - Intermediate code for object class MyClass

[0046] By examining the source code examples in Tables 7 through 9, and comparing them with the intermediate code examples in Tables 10 through 12, and especially the comments in Table 12, one of ordinary skill in the art will understand the transformations necessary for implementing a compiler according to the present invention.

[0047] While the present invention has been described in relation to an object-oriented environment, those skilled in the art will appreciate that these techniques may readily be applied to other programming paradigms.

55 Claims

Claims

1. A computer system having a processor, a memory, and a software entity comprising either a program body or one or more object classes, the software entity defining an overall functionality of a software application program, and

an aspect-oriented programming environment supporting an aspect that implements concerns that cross-cut the overall functionality of the software entity, the aspect comprising:

- 5 a) a cross-cut that comprises a point in the execution where cross-cutting behavior is to be included; and
 b) a cross-cut action comprising a piece of implementation associated with the cross-cut, the piece of implementation comprising computer-executable code that implements the cross-cutting behavior;
 c) wherein the cross-cut action transparently forces the cross-cutting behavior on the software entity.
- 10 2. A system according to claim 1, wherein the cross-cut defines a plurality of points in the computation associated with a plurality of places in the software entity.
- 15 3. A method for developing software in a computer system including an aspect-oriented programming environment supporting an aspect that implements concerns that cross-cut the overall functionality of the software entity, the method comprising:
 - 15 a) creating a software entity comprising either a program body or one or more object classes, the software entity defining an overall functionality of a software application program,
 b) creating an aspect which modifies at least one software entity by forcing behavior on the at least one software entity, wherein the modification is transparent to the at least one software entity;
 c) compiling the software entity along with the aspect to thereby force the behavior of the aspect onto the software entity.
 - 20 4. The method of claim 3, wherein the cross-cut defines a plurality of points in the computation associated with a plurality of places in the software entity.
 - 25 5. The method of claim 3 or claim 4; wherein creating an aspect comprises:
 - 25 1) creating a cross-cut defining a point in the computation where cross-cutting behavior is to be included; and
 2) creating a cross-cut action comprising a piece of implementation associated with the cross-cut, the piece of implementation comprising computer-executable code that implements the cross-cutting behavior.
 - 30 6. The method of any of claims 3 to 5, wherein the cross-cut is a concrete cross-cut that uniquely defines a point in the computation where cross-cutting behavior is to be included.
 - 35 7. The method of any of claims 3 to 6, wherein the computer system further includes an aspect library including a library aspect, the method comprising creating the cross-cut comprises creating a concrete aspect that reuses parts of an implementation of the library aspect.
 - 40 8. An aspect library for use in a computer system having a processor, memory, and an aspect-oriented operating environment, the aspect library for use in creating a software program, the software program comprising one or more software entities that define the overall functionality of the software program, each of the one or more software entities comprising one or more program bodies or one or more object classes, each of the one or more software entities comprising computer executable code which performs one or more functions, the aspect library comprising a plurality of aspects, each aspect comprising:
 - 45 a) a placeholder for specifying a cross-cut that comprises a point in the execution where cross-cutting behavior is to be included; and
 b) a cross-cut action comprising a piece of implementation associated with the cross-cut, the piece of implementation comprising computer-executable code that implements the cross-cutting behavior;
 c) wherein the cross-cut action transparently forces the cross-cutting behavior on the software entity.
 - 50 9. A computer-readable medium for use in a computer system having a processor, a display, and memory, the memory including an aspect-oriented operating environment, the computer-readable medium having an aspect library comprising a plurality of aspects, each aspect comprising:
 - 50 a) a placeholder for specifying a cross-cut that comprises a point in the execution where cross-cutting behavior is to be included; and
 b) a cross-cut action comprising a piece of implementation associated with the cross-cut, the piece of imple-

mentation comprising computer-executable code that implements the cross-cutting behavior;
c) wherein the cross-cut action transparently forces the cross-cutting behavior on the software entity.

10. A method for transferring a computer program product from one or more first computers to a second computer connected to the one or more first computers through a communications medium, the method comprising:

5 a) accessing, on the one or more first computers, an aspect library comprising a plurality of aspects, each aspect comprising:

10 1) a placeholder for specifying a cross-cut that comprises a point in the execution where cross-cutting behavior is to be included; and
2) a cross-cut action comprising a piece of implementation associated with the cross-cut, the piece of implementation comprising computer-executable code that implements the cross-cutting behavior;

15 3) wherein the cross-cut action transparently forces the cross-cutting behavior on the software entity; and
b) transferring the computer-executable instructions from the one or more first computers to the second computer.

20

25

30

35

40

45

50

55

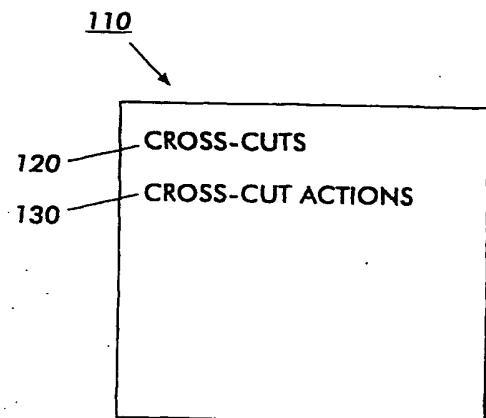


FIG. 1

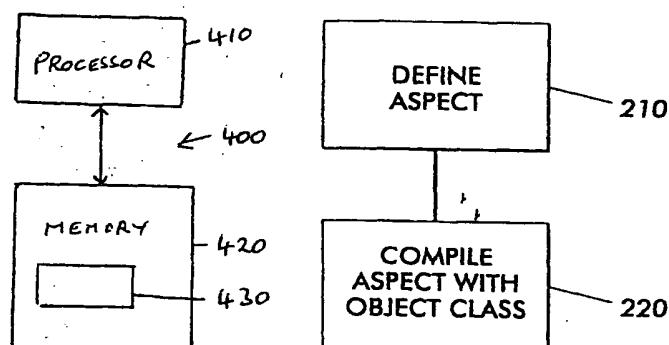


FIG. 4

FIG. 2

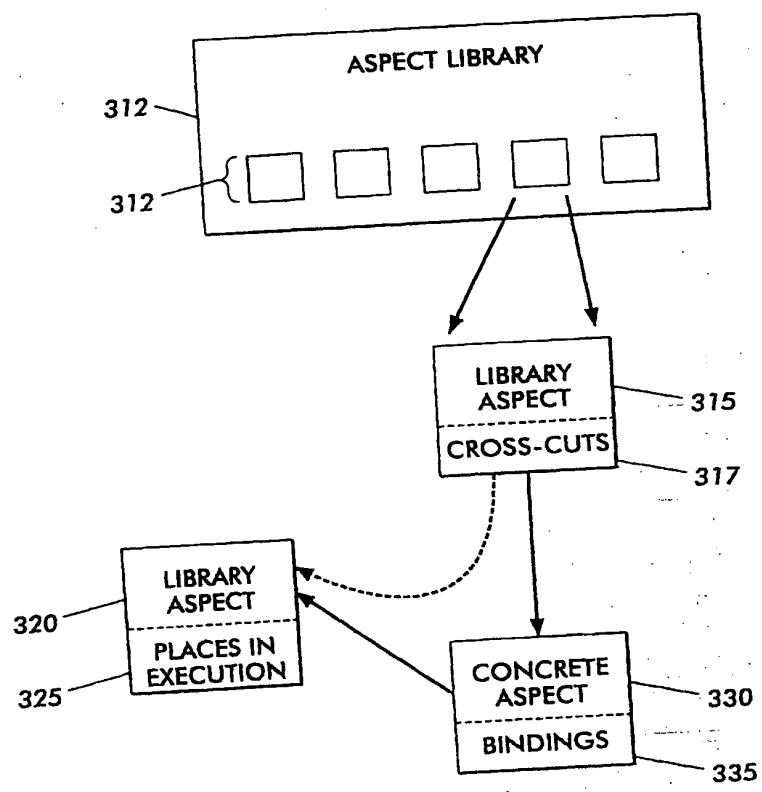


FIG. 3

THIS PAGE IS A
PRINT-OUT
(USPTO)